Data Warehousing and Data Mining

Lecture 3 Efficient Cube Computation

Acknowledgement: The Lecture Slides are adapted from the original slides of Han’s textbook.
Lecture Outline

• Types of cells

• Types of Cubes

• Efficient Computation of Data Cubes
  – Multiway Array Aggregation
  – BUC
  – Star Cubing
• A data cube is a **lattice** of cuboids.
  – Each cuboid represents a *group-by*.
  – The *base cuboid* is the least generalised of all the cuboids.
  – The *apex cuboid* is the most generalised of all the cuboids.

• **Operations**
  – Drill Down: move from the apex cuboid downward in the lattice.
  – Roll Up: move from the base cuboid upward in the lattice.

• **Commonly used measures include:**
  – `count()`, `sum()`, `min()`, `max()`
  – `average()`
Cell Types

- **Base cell**
  - A cell in the base cuboid

- **Aggregate cell**
  - A cell from a nonbase cuboid
  - It aggregates over one or more dimensions

- **Notations**
  - A cell is in general written as $a = (a_1, a_2, \ldots, a_n, measure)$
  - An aggregated dimension is indicated by an asterisk ($\ast$).
  - $a$ is an *m-dimensional* cell if exactly $m$ values in $\{a_1, a_2, \ldots, a_n\}$ are *not* $\ast$, $m (m \leq n)$. If $m = n$, then $a$ is a base cell.
  - What should $m$ be, if $a$ is an aggregate cell of the apex cuboid?
Example

• Consider a data cube with the dimensions *month*, *city*, and *customer group*, and the measure *sales*. What does the following notations mean?
  – (Jan, *, *, 2800) and (*, Chicago, *, 1200)
  – (Jan, *, Business, 150); and
  – (Jan, Chicago, Business, 45)
Ancestor and Descendant Cells

- An ancestor-descendant relationship may exist between cells.
- In a n-dimensional data cube, given two cells
  - an \( i - D \) cell \( a = (a_1, a_2, \ldots, a_i, \text{measure}_a) \)
  - a \( j - D \) cell \( b = (b_1, b_2, \ldots, b_j, \text{measure}_b) \)
  - \( a \) is an ancestor of \( b \), and \( b \) is a descendant of \( a \) if and only if
    - \( i < j \)
    - for \( 1 \leq k \leq n \), \( a_k = b_k \) whenever \( a_k \neq * \).
- In particular, cell \( a \) is called a parent of cell \( b \), and \( b \) is a child of \( a \), if and only if \( j = i + 1 \).

- Example:
  - 1-D cell \( a = (Jan,*,*,2800) \) and 2-D cell \( b = (Jan,*,Business,150) \) are ancestors of 3-D cell \( c = (Jan,Chicago,Business,45) \);
  - \( c \) is a descendant of both \( a \) and \( b \);
  - \( b \) is a parent of \( c \); and \( c \) is a child of \( b \).
Lecture Outline

- Types of cells
- Types of Cubes
- Efficient Computation of Data Cubes
  - Multiway Array Aggregation
  - BUC
  - Star Cubing
Types of cubes

- **Full cube**
  - All cells and cuboids are materialized.
  - All possible combination of dimensions and values (*prohibitively expensive*).
    - $2^n$ - exponential to the number of dimensions ($n$)
    - more cuboid if we consider concept hierarchies for each dimension
    - the size of the cuboid depends on the cardinality of its dimensions

- **Iceberg cube (tip of iceberg)**
  - Partial materialization
    - tradeoff between storage space and response time for OLAP.
    - Materializing only the cells in a cuboid whose measure value is above the minimum threshold.

- **Closed cube**
  - No ancestor cell is created if its measure is equal to that of its descendent cell.

- **Shell cube**
  - Only cuboids with limited number of dimensions are precomputed.
Closed Cube in more detail

• Suppose that there are 2 base cells for a database of 100 dimensions, denoted as
  \[\{(a_1, a_2, a_3, \ldots, a_{100}) : 10, (a_1, a_2, b_3, \ldots, b_{100}) : 10\}\], where each has a cell count of 10.

• Is iceberg cube good enough in sparse cases like this? How many cuboids?
  – Apex cuboid + 1-D cuboids + 2-D cuboids + … + 99-D cuboids
    \[\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{99} + 1 = 2^{100} - 1\]
  – Total number of aggregate cells is \(2^{101} - 6\)
  – Ignore all the aggregate cells that can be obtained by replacing constants with \(*\), only 3 cells really offer new information.
    \[\{(a_1, a_2, a_3, \ldots, a_{100}) : 10, (a_1, a_2, b_3, \ldots, b_{100}) : 10, (a_1, a_2, *, \ldots, *) : 20\}\]
Closed cell and closed cube

• Closed cell
  – A cell, c, is a closed cell if there exists no cell, d, such that d is a specialization (descendant) of cell c (i.e., where d is obtained by replacing * in c with a non-* value), and d has the same measure value as c.
  – No ancestor cell is created if its measure is equal to that of its descendent cell.

• Closed Cube
  – A closed cube is a data cube consisting of only closed cells.
Lecture Outline

- Types of cells
- Types of Cubes
- Efficient Computation of Data Cubes
  - Multiway Array Aggregation
  - BUC
  - Star Cubing
Multi-way Array Aggregation

- Used for MOLAP and full cube computation
- Array-based “bottom-up” algorithm
- Using multi-dimensional chunks
  - Direct array addressing
- Simultaneous aggregation on multiple dimensions
- Intermediate aggregate values are re-used for computing ancestor cuboids
- Cannot do Apriori pruning: No iceberg optimization
Example – Multi-way Array Aggregation
Example – cuboids to be computed

• The base cuboid,
  – denoted by ABC (from which all the other cuboids are directly or indirectly computed).
  – This cube is already computed and corresponds to the given 3-D array.

• The 2-D cuboids,
  – AB, AC, and BC, which respectively correspond to the group-by’s AB, AC, and BC.
  – These cuboids must be computed.

• The 1-D cuboids,
  – A, B, and C, which respectively correspond to the group-by’s A, B, and C.
  – These cuboids must be computed.

• The 0-D (apex) cuboid,
  – denoted by all, which corresponds to the group-by ()
  – That is, there is no group-by here.
  – This cuboid must be computed.
  – It consists of only one value.
  – If, say, the data cube measure is count, then the value to be computed is simply the total count of all the tuples in ABC.
• Assume the sizes of dimension, A, B, and C are 40, 400, 4000 respectively.

• Therefore AB is the smallest and BC is the largest 2-D planes

• If chunks are scanned as 1, 2, 3, ... then 156,000 memory units are needed
  (40*400+40*1000+100*1000)

• If chunks are scanned as 1, 17, 33, 49, 5, 21, 37 ... then 1,641,000 memory units are needed
  (aggregation ordering AB-AC-BC). Chunk memory units needed are (400*4000+40*1000+10*100)
What is the best traversing order?

Needs 156,000 Memory units

Needs 1,641,000 Memory units
Multi-way Array Aggregation

• **Method:** the planes should be sorted and computed according to their size in ascending order
  
  – **Idea:** keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane

• **Limitation of the method:** computing well only for a small number of dimensions
  
  – If there are a large number of dimensions, “top-down” computation and iceberg cube computation methods can be explored
Bottom-Up Computation (BUC)

- Bottom-up cube computation
  (Note: top-down in our view!)

- Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy $\text{min}\_\text{sup}$, its descendants can be pruned
  - If $\text{minsup} = 1 \Rightarrow$ compute full CUBE!

- No simultaneous aggregation
BUC Partitioning

- Usually, entire data set can’t fit in main memory
- Sort *distinct* values
  - partition into blocks that fit
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
    - Higher the cardinality-smaller the partitions-greater pruning opportunity
  - Collapsing duplicates
    - Can’t do holistic aggregates anymore!

Ideally the dimension with most discriminative, higher cardinality and having less skew is processed first.
BUC: Example \((\text{having count}(\ast) > 5)\)
BUC: Example (having count(*) > 5)
Till Now

- Aggregates simultaneously on multiple dimensions.
- Multiple cuboids can be computed simultaneously in one pass.
- Dynamic structure with simultaneous aggregation.

- Facilitates a-priori pruning.
- During partitioning, each partition’s count is compared with min sup. The recursion stops if the count does not satisfy min sup.

Top-Down Computation

Bottom-Up Computation
Star-cubing (aside)

- Bottom-up computation with top-down expansion of shared dimensions
- Base cuboid tree fragment
Star-tree construction and Star-Cubing

Base (Cuboid) Table: Before Star Reduction

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₁</td>
<td>d₁</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>c₄</td>
<td>d₃</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>a₂</td>
<td>b₂</td>
<td>c₂</td>
<td>d₂</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>a₂</td>
<td>b₄</td>
<td>c₃</td>
<td>d₄</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

One-Dimensional Aggregates

<table>
<thead>
<tr>
<th>Dimension</th>
<th>count = 1</th>
<th>count ≥ 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a₁(3), a₂(2)</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>b₂, b₃, b₄</td>
<td>b₁(2)</td>
</tr>
<tr>
<td>C</td>
<td>c₁, c₂, c₄</td>
<td>c₃(2)</td>
</tr>
<tr>
<td>D</td>
<td>d₁, d₂, d₃</td>
<td>d₄(2)</td>
</tr>
</tbody>
</table>

Compressed Base Table: After Star Reduction

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>a₁</td>
<td>b₁</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>a₁</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>a₂</td>
<td>*</td>
<td>c₃</td>
<td>d₄</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Aggregation stage one: processing the leftmost branch of the base tree.

Aggregation stage two: processing the second branch of the base tree.